PROPOSAL FOR RESEARCH ON INTERLISP AND NETWORK-BASED SYSTEMS

XEROX CORPORATION,
PALO ALTO, CALIFORNIA

26 OCTOBER 1976

365007 UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1 REPORT NUMBER: FINAL	2 GOVT ACCESSION NO.: N.A.	3 RECIPIENT'S CATALOG NUMBER: N.A.
4 TITLE (and Subtitle): PROPOSAL FOR RESEARCH ON INTERLISP AND NETWORK-BASED SYSTEMS		5 TYPE OF REPORT & PERIOD COVERED: FINAL 4/1/75 - 10/31/76
		6 PERFORMING ORG. REPORT NUMBER: N.A.
7 AUTHOR(s): W. Teitelman		8 CONTRACT OR GRANT NUMBER(s): N00014-75-C-0626
9 PERFORMING ORGANIZATION NAME AND ADDRESS: Xerox Corporation, Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA. 94304		10 PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: NR 049-372/12-3/75, Order. No. 2906, Code No. 5 D30
11 CONTROLLING OFFICE NAME AND ADDRESS: Office of Naval Research, Department of the Navy, Arlington, VA 22217		12 REPORT DATE: 10-26-76
		13 NUMBER OF PAGES: 12
14 MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): N.A.		15 SECURITY CLASS. (of this report): unclassified
		15a DECLASSIFICATION/DOWNGRADING SCHEDULE: N.A.

16 DISTRIBUTION STATEMENT (of this Report)

Unlimited Distribution

17 DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Unlimited Distribution

18 SUPPLEMENTARY NOTES

None

19 KEY WORDS (Continue on reverse side if necessary and identify by block number)

20 ABSTRACT (Continue on reverse side if necessary and identify by block number)

The contract covered by this annual report includes a variety of activities and services centering around the continued growth and well-being of INTERLISP, a large, interactive system widely used in the ARPA community for developing advanced and sophisticated computer-based systems.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102 LF 014 6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Sponsored by
Advanced Research Projects Agency
ARPA Order No. 2906

ARPA Order No. 2906
Program Code No. 5D30
Xerox Corporation
Effective Date: 1 April, 1975
Expiration Date: 31 October, 1976
Amount of Contract: $80,000

Contract Number: N00014-75-C-0626
Principal Investigator: W. Teitelman
(415) 494-4447
Title: Proposal for Research on
Interlisp and Network-Based Systems
Date of this report: October 26, 1976

Final Report

The contract covered by this annual report includes a variety of activities and services centering around the continued growth and well-being of INTERLISP, a large, interactive system widely used in the ARPA community for developing advanced and sophisticated computer-based systems. Perhaps the best way to report on the progress accomplished over the period of this contract is to present these activities in terms of the original work statement:

Statement of Work

a) Maintenance of the user portion of INTERLISP -- finding and fixing bugs reported by the ARPA community, and modifying or improving existing features.

b) Documentation -- producing and maintaining documentation for INTERLISP, including an on-line INTERLISP reference manual, and programs for convenient accessing of this manual, i.e. an information retrieval facility.

c) Distribution of new INTERLISP systems -- periodically bringing up and checking out new releases of the system, and distributing them to selected ARPA sites.

d) Coordination and user interaction -- insuring that the various users communities are kept informed of developments in INTERLISP, and that they have the opportunity to participate in and influence such developments. Also, insuring that new implementations of INTERLISP conform to INTERLISP standards.

"... maintenance of user portion of INTERLISP, finding and fixing bugs reported by the ARPA community, modifying or improving existing features, ..."

During the contract year, we received and responded to a host of reports of bugs, non-features, and suggestions for additions or changes, or simply requests for information, from twenty ARPA sites/user-communities: BBN-TENEXB, USC-ISIB, USC-ECL, USC-ISIC, SUMEX-AIM, BBN-TENEXD, SRI-AI, BBN-TENEXA, SU-AI, MIT-AI, HARV-10, BBN-TENEXC, SCI-TENEX, USC-ISID, CMU-10A, MIT-ML,

UTAH-10, RUTGERS-10, MIT-MC, OFFICE-1 The following examples are typical of the range of these interactions:

A user at ISI (Goldman) reported a bug in the translation of iterative statements which was corrected.

A user at ISI (Vittal) suggested an extension to the iterative statement in the form of a REPEATWHILE and REPEATUNTIL operator. This extension was of general value, and was implemented.

A user at SU-AI (Davis) found a problem with the translation of the subset operator. The bug was eliminated.

A user at SUMEX (Lederberg) encountered a problem which was traced to the facility for reusing the structure associated with large numbers. This information was relayed to BBN and the problem was fixed.

A user at SU-AI (Davis) found a bug in the history facility of the programmer's assistant, which was fixed.

A user at SU-AI (Gordon) working on the mathematical semantics of advanced LISP systems requested information on what would be the spaghetti-stack primitives in INTERLISP, in order to determine if his semantic domains were adequate for representing the denotations of environment manipulating operations. We supplied him with a draft form of the spaghetti stack documentation and answered some additional questions.

A user at BBN (Lewis) found a bug relating to nested F/L operators in DWIM.

In response to a request from BBN (Lewis), the iterative statement of DWIM/CLISP was extended to permit the user to specify an expression which *computed* the expansion of an operator. This significantly increased the range of applications of the iterative statement, and enabled a number of projects to take advantage of the benefits of increased clarity of code that the iterative statement provides.

In response to a request from BBN (Bates), we provided a description of a PARC developed package (Teitelman) for allowing the user to load a large, heavily commented, symbolic program, without actually having to load, and pay the storage costs for all of the comments. As a result of its usefulness to the BBN project, this package was documented and included in the INTERLISP system.

A user at SUMEX (Van Melle) requested a way of disabling certain CLISP operators with respect to CLISPIFYING, i.e. converting LISP programs to CLISP, while leaving active the inverse transformations from CLISP to LISP. This facility did not exist, and so was added.

A user at ISI (Ryland) discovered a flaw in the DWIM scheme for recovery following an error in certain situations. DWIM was revised and extended to cover this case.

In response to several requests, a facility developed at PARC (Teitelman) for automatically marking those portions of a programs listing corresponding to changes was made a part of the standard INTERLISP system and released to the ARPA community at large.

A user at USC-ECL (Britt) discovered and reported a serious problem with interrupt characters, which was forwarded to BBN and repaired.

A user at SRI (Paxton) discovered a problem with the user interface with the compiler, which was repaired by PARC.

BBN (Lewis) requested a way of augmenting the English-to-LISP transformations in CLISP, e.g. to be able to tell CLISP that a phrase such as "X IS THE BEGINNING OF AN ARRAY" should translate into the LISP expression (AND (ARRAYP X) (ARRAYBEG X)). This facility was added.

ISI (Yonke) reported that the greeting facility in INTERLISP malfunctioned when the number of directories in the host TENEX was increased beyond 511. This was repaired.

SCI-TENEX (Holsworth) requested a version of INTERLISP. We supplied same and helped them with the necessary initialization procedures.

PARC (J Moore) discovered a bug in the compiler relating to the use of free variables, which was reported to BBN and fixed.

Charles Vollum, of the Oregon Museum of Science and Industry, contacted PARC about an implementation of INTERLISP on the PDP-11 that he was working on. We sent him a copy of the VM specification, and put him in touch with the people at BBN who were interested in the same project.

Mark James, of the University of California at Irvine, contacted PARC about implementations of INTERLISP for the CDC 3300, and for TOPS10, which he was working on. We gave him a complimentary account at PARC so that he could experiment with INTERLISP, and interact with us via the message facility about ideas that he had, and problems he encountered. He took advantage of this invitation and the PARC facility frequently.

To facilitate the expected frequent release of new and incompatible versions of the INTERLISP-10 compiler, BBN (Hartley) requested installation of a facility for automatically detecting when a user attempted to load a file compiled with an incompatible (older or newer) compiler.

A user at SUMEX (Van Melle) found a subtle but potentially serious problem with the translation of iterative statements by DWIM. This bug was fixed

A user at PARC (Teitelman) encountered and isolated several bugs relating to applications in which user programs performed their own input operations directly, instead of by using the INTERLISP READ program.

A user at PARC found a bug in the garbage collector, which was reported to BBN and fixed.

Another user at PARC found a bug in the δ compiler, which was also reported to BBN and fixed.

Jaak Urmi, of Uppsala University in Sweden, who is implementing an INTERLISP for the IBM 370, visited PARC and engaged in a lengthy discussion about the INTERLISP Virtual Machine with several PARC scientists (Bobrow, Deutsch, Moore, Teitelman). The result was a much greater degree of agreement between his implementation on INTERLISP as it is currently implemented. Should this implementation become available in the United States, transporting existing INTERLISP programs to the 370 implementation will be much easier.

At the suggestion of a BBN user, a facility was provided whereby the user could release (some of) the storage required for facilities that he did not use, or was not using at the time. Previously, the burden was on the user to know just how to recover this storage. The new facility,

GAINSPACE, walked the user through a menu of options, querying him about each facility and allowing him to select which to discard and which to retain. GAINSPACE is important to users developing systems with very large data bases.

There were several inquiries about INTERLISP for the new DEC KL-10. We acted as intermediary for the various sites involved. There was also continued discussion with Mark James, University of California, Irvine, about an implementation of INTERLISP for TOPS-10.

A user at SRI-AI (Boyer) found some problems with the new facility for user defined iterative operators, which was then repaired.

A user at SRI-AI (Spitzen) requested information about an INTERLISP to MACLISP translator. We helped him with the problem of making this transformation.

A user at PARC found a bug in the compiler relating to compiling of non-local GOTO's. This bug was reported to BBN, and fixed in about a week. At that point, a user at SRI encountered the same bug. As a result of a PARC user having previously encountered the same bug, we were able to immediately supply the SRI user with the necessary fix, otherwise he would have had to wait. *The benefit to other ARPA sites of having PARC continuously exercising INTERLISP releases is an important one*: it drastically reduces the number of bugs encountered by other ARPA users, and therefore reduces the disruption to their work.

A user at ISIB (Greenfeld) found a bug in the ASKUSER package and also suggested some improvements and extensions which were implemented.

We had several interactions several times with Mark James, University of California at Irvine, on his implementation of INTERLISP for the CDC 3300, to insure that it would be compatible with INTERLISP-10.

Following BBN's release of a new compiler designed to produce more efficient code, several subtle bugs were encountered and isolated by PARC users.

A user at ISIB (Yonke) reported some bugs in DWIM which were fixed

A user at SUMEX (Van Melle) reported a bug in PRETTYPRINT relating to the new facility for optionally keeping the text of comments on files when the source for the program was loaded. (This facility had been included in the INTERLISP system at BBN's request, as described in the previous quarterly management report.)

A user at CMU-10A (Clark) needed some help in reviving some old versions of INTERLISP in order to perform some measurements on some old sysouts. We provided these programs from our archives.

A user at ISIB (Greenfeld) requested assistance in converting some programs from MACLISP to INTERLISP. We helped him in this project.

We provided a user at BBN (R Bobrow) the sources for a parsing program developed at PARC (Kaplan) for use in conjunction with the intelligent terminal project. We also provided him with assistance on the use of the parser, and made some changes that he requested.

Several extensions to the record package requested by users at ISI were provided, including a synonym facility, a way of computing the field names of a given record, and several new record types. Masterscope was extended to know about records.

A user at ISIB (Srinivasan) was experiencing a great deal of difficulty getting his program to load into a stripped down version of INTERLISP (it was too large to load into the full INTERLISP). We provided him with assistance and suggestions over a period of a few weeks until he was successful.

A user at SUMEX (Stefik) needed a facility which would make a copy of any INTERLISP data structure, including structures that contained user defined datatypes, and were potentially circular or reentrant. This facility was provided (Masinter).

A user at SUMEX (Van Melle) found a problem with the ordering of tests in the translation of iterative statements.

A user at BBNA (Harris) suggested an improvement to the algorithm used in translating F/L expressions. This improvement was implemented.

A user at SRI-AI (Fikes) requested a way of selectively turning off various CLISP operators that was currently not possible. This addition was performed.

"... modifying or improving existing features."

During the contract year, there were a number of significant extensions or additions to INTERLISP contributed by PARC scientists under PARC supported research. While these do not fall directly under the aegis of the ARPA contract, nevertheless they are a logical outgrowth of PARC's involvement with INTERLISP and the ARPA community, and significantly benefit both.

A fast string searching algorithm was invented by Boyer (SRI) and Moore (PARC), and implemented in INTERLISP by Moore. The new procedure was generally 10 to 50 times faster than the existing routines for searching for strings in a file, and is now heavily used for applications calling for extensive file searches, e.g. the on-line documentation of the INTERLISP system.

An extremely general user interaction package, ASKUSER, was designed and implemented at PARC (Teitelman). A single call to ASKUSER can be used to specify the protocol for an entire sequence of interactions. At each point, ASKSUSER can prompt the user, give him a list of acceptable responses, or allow him to start over. For example, the TENEX EXEC interactions with the user could be entirely handled with a single call to ASKUSER.

The File Package was extended to inform the user when functions, records, or other entities were changed, and therefore needed to be written out, but the corresponding entity was not associated with any particular file. This situation frequently occurs when a user is in the early stages of developing a system. If the user forgets to associate the new entities he defines with a symbolic file, they would not be dumped, and would therefore be lost when the user reloaded in a new system. The extension to the file package causes the user to be reminded, and allows him to specify interactively and conveniently on what files the new entities are to be placed.

The iterative statement construct of CLISP was extended to permit the user to define his own iterative operators. This extension enables the user to take advantage of the clarity of syntex provided by the iterative statement in expressing operations not built into the iterative statement. For example, the user could define AVERAGE as an operator with an appropriate initialization: FIRST SUM ← 0 N ← 0, main step: SUM←SUM+BODY, N←N+1, and a finalization which computed the average: FINALLY RETURN SUM/N. He could then write iterative statements such as FOR X IN Y AVERAGE X WHEN X GT 0, or FOR I FROM 1 TO 20 AVERAGE I+2 WHEN (predicate I).

A coroutine and generator package was designed and implemented (Bobrow and Kaplan) which greatly simplified the use of the new spaghetti stack capability in INTERLISP. SRI (Fikes) was invited to use the package and made a number of valuable suggestions. BBN (Lewis) provided some important compiler interfaces necessary for efficiency. Virtually all applications involving switching of contexts now use this package, e.g. the ARPA Decision Aids Project and the Speech Understanding Project (SRI) depend heavily on it.

An already existing facility was extended and generalized to permit programs to protect themselves against aborts, such that any undoable changes executed by the program would automatically be undone if the user typed the abort key (PARC, Teitelman).

A Hashfile package was designed and implemented (Masinter) for creating and accessing a large database on a file. The databases addressed by this package are much larger than could be stored in the user virtual address space. The Hashfile package provides a symbol-table facility with a large capacity with a relatively small additional cost in run-time over using INTERLISP atoms. It was subsequenlty extended (Kaplan) to enable complete list-processing on a file. The Hashfile package is now heavily used by the medical diagnosis project (MYCIN) at SUMEX, as well as various projects at BBN.

The design and implementation of Masterscope (Masinter) was completed. Masterscope is an interactive program for analyzing and cross referencing user programs. It contains facilities for analyzing user programs to determine what other programs are called, how and where variables are bound, set, or referenced, as well as which programs use particular record declarations. Masterscope is able to analyze definitions directly from a file as well as in core definitions. Masterscope is interfaced with the INTERLISP editor so that when a program is edited, Masterscope automatically notes that it needs to be reanalyzed. Masterscope is also interfaced with the INTERLISP file package, so that in the case that the source code for a compiled program has not been loaded, Masterscope can automatically retrieve it and analyze it on the file. Masterscope was provided with a flexible English-like comand language, so that the user can express complex operations in a straightforward fashion, e.g. WHAT FUNCTIONS USE X AND CALL ANY FUNCTION THAT BINDS X, SHOW WHERE ANY FUNCTION USES FOO AS A RECORD FIELD, etc. A facility for automatically checking a set of block declarations for consistency as well as typical errors, using heuristics gathered from various users, was incorporated into Masterscope. This facility has turned out to be extremely useful for working with large systems of programs.

The programmers assistant was extended to handle reexecution of an event or events for a specified number of iterations, or while a particular condition was true, e.g. REDO event WHILE X IS LESS THAN 10.

"Maintaining documentation for INTERLISP, producing new documentation where necessary, said documentation to be on-line, machine readable."

We revised, updated, and totally reformatted the INTERLISP manual. The reformatting was designed to take advantage of the multiple fonts provided by our in-house printing facility. By use of smaller fonts for footnotes, as well as going to single space, we were able to reduce the size of the manual considerably, while at the same time improving its readability.

The entire preparation of the manual was automated using the PUB facility. Our conventions were embodied in special PUB macros developed for the purpose of insuring consistency of format, both for aesthetic reasons as well as to enable processing of the text by HELPSYS, the on-line documentation system simultaneously being developed (see below). This approach also made feasible the flagging (with a special character in the margin) of any material that was changed or added. Thus, users already familiar with the INTERLISP system could skim the new manual and quickly see what was different.

The new manual provided documentation for a number of important facilities that had appeared since the publication of the last manual in October, 1974, including: the spaghetti stack capability, the coroutine and generator package, user data types, Masterscope, ASKUSER, user interrupt characters. The new manual also incorporated a number of suggestions made by INTERLISP users for making the manual clearer and easier to use. All told there were changes to 174 pages (of approximately 600 total pages), a testimonial to the evolving nature of the INTERLISP system. 750 copies were printed and are being distributed.

"...(producing) programs for convenient accessing of this manual, i.e. an information retrieval facility."

We implemented Helpsys, an on-line documentation system that uses the INTERLISP Reference Manual as a data base. Using Helpsys, the user can ask questions about topics in the INTERLISP manual, using a constrained but English-like command language, e.g. "TELL ME ABOUT PRETTYPRINT," "WHAT ARE THE ARGUMENTS TO COMPILE." Where there are several subcategories of the indicated subject, Helpsys will allow the user to specify more precisely what it is he wants to know. For example, if the user asked "TELL ME ABOUT PROP," Helpsys would repond "(1) Prettydef command, (2) reference typed by editor, (3) error message, (4) DEFINE {cross referenced subject}, Which one?" Helpsys knows about descriptions of function definitions, system flags, editor commands, break commands, etc. as well as general topic references, e.g. "TELL ME ABOUT MAKING A SYMBOLIC FILE," "TELL ME ABOUT COROUTINES AND GENERATORS." For each request, Helpsys presents a specified amount of text (for display terminals, one screenful). The user can then continue to browse from that point forward or backward in the manual, or give a new request. When given an unrecognized topic, Helpsys performs spelling correction as well as "wildcard" completion. For example, if the user has forgotten the name of a particular flag in the system, but knows it begins with CLISP and ends with FLG, he can say: "TELL ME ABOUT CLISPSFLG," whereupon Helpsys will provide him with a list of categories which match with CLISPSFLG.

Helpsys is now a part of the standardly released INTERLISP system, and the Helpsys database, i.e. a specially formatted representation of the INTERLISP manual, has been distributed to various ARPA sites. We expect that Helpsys will make the large amount of information contained in the INTERLISP Manual more readily available to on-line users.

"Periodically bring up and checking out new releases of the system, and distributing them to selected ARPA sites."

During the early part of the contract year, a great deal of activity centered around shaking down the newly released (by BBN) spaghetti stack system. The spaghetti stack capability had been under development and eagerly awaited for several years. It allowed programs to construct, maintain, and switch back and forth between several different environments simultaneously, and was absolutely essential for many of the advanced application systems, such as command and control, now being developed. As is usually the case with such a major change in a large system, a great number of problems were encountered by users at PARC. Frequently these problems were of the nature as to totally destroy the program that was being run when they were encountered. Fortunately, we were able to interact quickly with BBN implementors via the ARPA net, and the problem would usually be fixed within a day or so. At that point, we would reload the system and continue trying to break it. During some phases of this effort, this was almost a daily process: during the first quarter alone, BBN released 30 new versions of the basic system, most in

response to problems encountered by PARC users. This process required much patience on the part of the users at PARC who were the guinea pigs for this experimental system. However, as a result of PARC users serving this function, the number of bugs that were subsequently encountered by users at other ARPA sites, and the consequent disruption of their work, were greatly reduced. Finally, a fairly stable and robust system emerged and was released to the ARPA community.

In addition to releasing the spaghetti stack system to ARPA sites, we produced and provided documentation for the new spaghetti stack system. In addition to providing reference material that was later included in the new INTERLISP manual, this documentation was designed to aid users in making the conversion from the pre-spaghetti stack system to the new system. It contained the results of our six months of experience with the system: what things to look for in the users code, how to rewrite them, pitfalls to avoid, subtle efficiency issues, etc. In addition to the documentation, W. Teitelman visited SRI and ISI and presented seminars on the new system and how to convert to it.

A principal effort during the latter part of the contract year involved shaking down and checking out the newly released (by BBN) version of the basic INTERLISP system that employed a new scheme for representing the values of variables called "shallow binding."

"Deep binding," the scheme employed for binding variables previously in the INTERLISP, involved storing the name and value of the variable on a stack. Accessing the value of a variable, either to obtain its value or to assign it a new value, thus involved searching up the stack looking for its binding. The time required was proportional to the distance on the stack to its most recent binding.

With shallow binding, a variable is bound by storing its value in a special cell associated with the variable called its value cell. The current, i.e. old, value of the variable is then stored on the stack. Thus accessing a variable requires no searching at all, and simply involves accessing its value cell. However, binding and unbinding are more expensive operations than under the deep binding scheme. Furthermore, spaghetti stack operations, i.e. those that involve switching contexts, are also more expensive. However, preliminary measurements undertaken by BBN and PARC indicated that on the balance, shallow binding would be more efficient than deep binding, since a considerable portion of the running time of a typical program was spent in looking up free variables. Furthermore, shallow binding had the benefit that programs that were run interpretively, as is usually the case when they are under development, would run much faster than with deep binding. (Normally, efficiency arguments are considered with respect to the finished product. However, this overlooks the fact a significant portion of the useage of INTERLISP is in developing and experimenting with systems. Thus, improvements in the running of interpretive programs are important.) Shallow binding also provided the benefit that large programs would not require the esoteric tuning and reconfiguring to obtain maximal efficiency that they currently did under deep binding.

Converting to shallow binding required an intensive amount of effort by BBN over a period of many months. The first shallow binding system was released to PARC on January 16. At that point, we began converting the INTERLISP library to conform to the new binding scheme. Some of the difficulties that we had in the this conversion process suggested changes which would facilitate the conversion process for less experienced users. In addition, as is always the case with so basic and pervasive a change to a large system, we found a number of serious bugs both in the basic system released by BBN and the corresponding compiler. As a result, our interactions with BBN over the network were many and varied. For example, during one three day period in January, 137 messages were exchanged between PARC and BBN. Each time a bug was encountered, we informed BBN (via the message facility). When a new basic system was available which repaired this particular problem, we would then reload and continue with the conversion and check out. This iteration frequently operated on a daily basis. For example, from January 16

when the shallow binding system was first released, through the end of the quarter, BBN released 34 different versions of the basic INTERLISP system, and 12 versions of the compiler. Each new version required reloading the entire INTERLISP system. In addition, some bugs required completely recompiling the INTERLISP library. As a result, enormous quantities of CPU time were expended. For example, in January alone, we used thirty eight hours of CPU time in this process, almost twice as much as the next biggest user at PARC. (Actually, this computer time was donated, since ARPA does not pay for the computer time used under this contract) When we finally were successful in converting the entire INTERLISP system and getting it to run, the new experimental system was released to users here at PARC, and they in turn began finding more bugs.

Meanwhile, we were dismayed by our initial timings on the new system. At first, programs ran slower in the new shallow binding system than under deep binding. A lengthy sequence of interactions with BBN during which we timed and made adjustments and timed some more finally led to discovery of the critical elements in the system, and in producing a version of the shallow binding system which achieved our expectations. It would be an understatement to observe that the checkout and tuning of the shallow binding system would have been extremely difficult, and taken much longer, had we not been able to interact with BBN rapidly, as was possible via the ARPA network.

"Insuring that new implementions of INTERLISP conform to INTERLISP standards."

In order to provide some uniformity for the various implementions of INTERLISP planned or under developement, we decided to try to define and set down on paper the specifications for an INTERLISP Virtual Machine. The virtual machine (or simply VM) is the underlying LISP system on top of which is built the large and sophisticated collection of user support facilities which make up the bulk of INTERLISP (such as DWIM, the programmer's assistant, etc.). Thus, if VM LISP is implemented on some machine, the rest of INTERLISP can be obtained from publicly available files and simply loaded and run. The document which we prepared is essentially an implementor's specification of VM LISP for INTERLISP. It makes explicit (for the first time anywhere) exactly what an implementor has to provide in order to satisfy the assumptions of the large collection of INTERLISP programs in existence. For example, INTERLISP assumes the existence of entities called atoms. Associated with each atom is a datum called its value cell. Values are obtained via the function GETTOPVAL, and set via the function SETTOPVAL. The implementor is free to represent atoms in any fashion that he chooses, so long as he provides the functions GETTOPVAL and SETTOPVAL to interface the user's programs with his particular implementation.

The VM project produced an implementation-independent set of primitives for performing basic operations such as input/output, manipulating the stack, generating and handling errors, etc. As the project progressed, we interacted increasingly with BBN, who was responsible for maintaining that part of INTERLISP-10 which corresponded to the VM. We tried to reach agreement with BBN on anything in the VM specification that would require a change to the INTERLISP-10 implementation, because BBN would have to make that change. We felt that any primitives defined in the VM must be provided in the VM implementation for INTERLISP-10, the most widely used implementation, or else the VM specification would just be an academic exercise. This constraint injected a much needed pragmatic note into the effort.

Wherever we found implementation *dependent* primitives, and reached agreement with BBN on how to express and provide these primitives in an implementation *independent* manner, we converted the corresponding INTERLISP programs to use the new primitives. It turned out that many INTERLISP programs, including a large portion of the INTERLISP system itself, contained many hidden assumptions about the implementation, in this case INTERLISP-10 on Tenex. To this

extent, they were machine dependent, and might not run on (or at least require some converting to run on) some of the implementations of INTERLISP for other machines that were currently in progress (not to mention the fact that the implementors had no precise guide of what was expected of their implementations.) As a result of the VM project, most of the INTERLISP system is now written in terms of implementation independent primitives described in the Virtual Machine Specification. Some parts of the INTERLISP system remained implementation dependent, such as those programs which provided direct access to Tenex capabilities which were not specified in the VM because they might not be available in other implementations. These programs were separated out and put in different files. Thus, at the end of the project, we had a set of source files describing the bulk of the INTERLISP system in an implementation independent manner.

This claim was then successfully put to a test by a project underway at PARC to implement INTERLISP on a Xerox mini-computer. If our supposedly implementation independent code could in fact be transported from the PDP-10 to this mini-computer and would in fact run, we would have some evidence that these programs were machine independent, and that the specifications in the VM were sufficient to guarantee transportability. We were in fact able to perform this exercise. While this implementation of INTERLISP is not itself an ARPA project, the results of these efforts will greatly benefit BBN's implementation of INTERLISP for the PDP-11, which is an ARPA project.

The Interlisp Virtual Machine Specification Document is included (under separate cover) as part of this final report.